Homework 8 v1  
EECS150, Fall 2012  
Due Friday, 6 pm, November 2, outside 125 Cory

1. Recreate figure 7.49 assuming that an interrupt request (IRQ) is asserted during cycle 1, and on the rising edge beginning cycle 2 the address 0xc0000180 is latched into imem. The interrupt routine starts with

```
mfc0 $k0, Cause;  
mfc0 $k1 Status  
```

and ends with

```
jr $k0 ; return to next instruction  
mtc0 $k1, Status ; reset the IE bit in the branch delay slot  
```

a. Use your 3 stage pipeline implementation rather than the 5 stage in the figure.

b. Why is no forwarding needed after the add and during the two mfc0 instructions?

c. Why is no forwarding needed after the mtc0 and during the and or instructions?

d. Assuming all ISRs begin and end this way, and assuming that the normal code never uses the kernel registers $k0 and $k1, will there ever be data hazards due to interrupts in your 3-stage pipeline?

2. Repeat 1abcd for Figure 7.51 assuming that the IRQ is again asserted during cycle 1, and taken on the rising edge of cycle 2. Assume a load delay slot is used.

3. Repeat 1abcd for Figure 7.54 when the branch is

a. taken and  
b. not taken,  
again assuming the IRQ is asserted during cycle 1, and assuming a branch delay slot is used. You decide if any delay in the interrupt being taken is necessary.

4. Figure 7.63 shows a single-cycle datapath implementing mfc0 with Cause and EPC registers, as well as loading the PC with the ISR address and loading EPC with the exception return address. Draw the same figure for your 3-stage pipeline MIPS in a format similar to figure 7.58. You don’t need to show the details of your MIPS, except where they are impacted by the mfc0 instruction. In particular:

a. label the pipelined versions of any new datapath components (c0 becomes c0_E and c0_W) and control signals (EPCwrite, IntCause, CauseWrite)

b. show where the rs, rt, and rd portions of the instruction word go (Table B.1 explains the format and functionality of the mfc0 and mtc0 instructions)

5. Redraw the figure above, adding the Status register, Count register, Compare register, and logic required to implement mtc0. In addition, show any new hardware necessary to generate the interrupt request signals Timer_IRQ, RTC_IRQ, and UART_IRQ, and where those signals go.